VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

HLUBOKÉ POSILOVANÁ UČENÍ A ŘEŠENÍ POHYBU ROBOTU TYPU HAD
DEEP REINFORCEMENT LEARNING AND SNAKE-LIKE ROBOT LOCOMOTION DESIGN

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jakub Kočí**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Radomil Matoušek, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Jakub Kočí**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Hluboké posilovaná učení a řešení pohybu robotu typu had

Stručná charakteristika problematiky úkolu:

Úkolem práce je předložit návrh a simulační ověření řídicího systému definovaného robotu typu had. Návrh řízení bude založen na hlubokém posilovaném učení. Navrhnuté řešení bude umožňovat robotu efektivní pohyb v přímém směru (undulatory locomotion) a bočním směru (sidewinding), včetně řešení pohybu v prostředí s překážkou.

Cíle diplomové práce:

- Rešerše RL algoritmů (DQN, DDPG, A2C, A3C).
- Implementace modelu.
- Optimalizace modelu RL algoritmem.
- Verifikační experiment a vyhodnocení.

Seznam doporučené literatury:

VONDRÁK, Ivo. Umělá inteligence a neuronové sítě. 3. vyd. Ostrava: VŠB - Technická univerzita Ostrava, 2009. ISBN 978-80-248-1981-5.

ZHANG, Yunong, Dechao CHEN a Chengxu YE. Toward deep neural networks: WASD neuronet models, algorithms, and applications. Boca Raton, Florida: CRC Press, [2019].

Abstrakt

Tato diplomová práce se zabývá použitím posilovaného učení pro úkoly hlubokého učení. V teoretické části je rozebrán potřebný základ k neuronovým sítím a posilovanému učení. Práce popisuje teoretický model posilovaného učení - Markovovské procesy, na konvenčních algoritmech ukazuje některé zajímavé techniky a v rešeršní části ukazuje některé z používaných algoritmů hlubokého posilovaného učení. Praktická část práce se skládá z vlastního modelu robotu a prostředí a z vlastního systému posilovaného učení.

Abstract

This master thesis is discussing application of reinforcement learning in deep learning tasks. In theoretical part, basics about artificial neural networks and reinforcement learning. The thesis describes theoretical model of reinforcement learning process - Markov processes. Some interesting techniques are shown on conventional reinforcement learning algorithms. Some of widely used deep reinforcement learning algorithms are described here as well. Practical part consist of implementing model of robot and it's environment and of the deep reinforcement learning system itself.

Klíčová slova

Posilované učení, hluboké posilované učení, hluboké učení, neuronové sítě, robot typu had, Python, CoppeliaSim, BlueZero

Keywords

Reinforcement learning, deep reinforcement learning, reinforcement learning, neural networks, snake-like robot, Python, CoppeliaSim, BlueZero

KOČÍ, Jakub. Hluboké posilovaná učení a řešení pohybu robotu typu had. Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/125129. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Radomil Matoušek.

Čestné prohlášení

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením doc. Ing. Radomila Matouška, Ph.D a s použitím literatury uvedené v seznamu literatury.

V Mostě dne 26. června 2020 Jakub Kočí, v.r.

Rád bych poděkoval doc. Ing. Radomilovi Matouškovi, Ph.D, který mi umožnil provést výpočty související s touto prací na počítačích v majetku ústavu.

Obsah

1 Úvod

V technické praxi se používá množství různých druhů robotů. Od létajících robotů - dronů přes roboty kráčející, ať už dvou- nebo vícenohé, k robotům kolovým. Roboty typu had jsou konstruovány buď jako kolové (s pasivními koly) nebo v „bezkončetinovém“ provedení, pohyb robotu zajišťuje vlastní dělení robotu na segmenty. Mají výhody při použití ve stísněných, neznámých prostorách: lze je konstruovat dostatečně malé, nehrozí jim výjimečné situace typu pád. Pokud jsou konstrukčně uzpůsobeny, mohou zdolávat i vertikální překážky - limitujícím rozměrem je samotná délka robotu.

Rozmach použití těchto „sofistikovanějších“ typů robotů přinesly až objevy v oblasti umělé inteligence a strojového učení - popsat pohyb robotu v explicitním vztahu je obtížné, a tak se řízení opírá především o učení robotu jako autonomního agenta. Stavový a akční prostor agenta je tak velký, že pro řízení není zbytí než použít buď velmi hrubou diskretizaci prostoru, anebo funkční aproximátory, jako třeba právě neuronové sítě.

Učení neuronových sítí na základě jejich vlastní interakce s prostředím zažívá v současnosti období prudkého růstu. Od objevu prvního „moderního“ algoritmu hlubokého posilovaného učení - publikování algoritmu DQN ([32]) uběhlo teprve 5 let. Obor hlubokého posilovaného učení je stále nový a dynamicky se rozvíjející.

2 Neuronové sítě

Neuronové sítě patří v oblasti strojové inteligence mezi biologicky inspirované procesy. V tomto případě je použito mozkové buňky - neuronu. Základní idea je prostá: mozek je složen z množství neuronů. Každý neuron sbírá informace ze synapsí. Na základě těchto vstupů rozhoduje tělo neuronu (*soma*), jestli neuron bude nebo nebude aktivován. Výstup neuronu se potom rozšíří přes axon na synapse dalších neuronů.

Umělé neuronové sítě umožňují řešení silně nelineárních problémů. Dokáží zevšeobecňovat, a učit se. Používají se jako univerzální funkční aproximátory, na úlohy regrese, klasifikace, predikce, řízení a další.

2.1 MCP neuron

[1, 2]

McCullochův-Pittsův model neuronu (1943) používá binární logiku - vstupy i výstup neuronu jsou pravdivostní hodnoty. Neuron bude dávat výstup 1, pokud jeho vnitřní potenciál přesáhne prahovou hodnotu (*threshold*), v ostatních případech je výstup 0. Potenciál se získá jako suma součinů vstupu a jemu příslušné váhy. Váhy jsou parametry neuronu a jsou to právě ony, ve kterých se ukládá naučená informace.

$$y = \begin{cases} 1 & p \geq t \\ 0 & p < t \end{cases} \tag{2.1}$$

$$p\left(x\right) = \sum_{i=1}^{n} x_i w_i \tag{2.2}$$

2.2 Perceptron

Perceptron (Frank Rosenblatt, 1957) je označení jednovrstvé neuronové sítě, která se skládá z jediného neuronu. Mapuje vstup reálného vektoru na binární (později též reálný) výstup. Jeho funkce je obdobná jako v případě MCP neuronu (rovnice (2.1) a (2.2)) [1, 3, 4].

$$y = S\left(p\right) \tag{2.3}$$

$$p\left(x\right) = \sum_{i=1}^{n} x_i w_i - t \tag{2.4}$$

Z rovnice (2.4) vidíme, že práh t lze vyjádřit jako by se jednalo o váhu w_0, jejíž vstup x_0 je vždy -1.

$$p\left(x\right) = \sum_{i=0}^{n} x_i w_i \tag{2.5}$$

Rovnice (2.5) má tvar vhodný pro kompaktní vyjádření perceptronového učícího pravidla.

Z geometrického pohledu perceptron rozděluje N-dimenzionální prostor nadrovinou na dva poloprostory. Učení spočívá v úpravě parametrů nadroviny tak, aby perceptron reagoval na maximální možné množství vstupů korektně.

2.2.1 Perceptronové učící pravidlo

Perceptron byl již v době svého vzniku doplněn učícím pravidlem, díky kterému se mohlo použít učení s učitelem pro trénink perceptronu. Jednovrstvá perceptronová síť ale dokáže řešit jen lineárně separovatelné problémy, což vedlo ke kritice tohoto modelu ze strany Marvina Minskyho a Seymoura Paperta ([5]), kteří se domnívali, že perceptronový model má velmi omezené použití, neboť nedokáže vyřešit např. logickou funkci XOR. Tehdy již bylo známo, že takové problémy lze řešit vícevrstvou neuronovou sítí. Nebyl však znám žádný algoritmus, který

by byl schopen vícevrstvý perceptron učit[1]. Názor, že algoritmus, který by dokázal učit vícevrstvé neuronové sítě, nebude objeven, v kombinaci s nedostatečně výkonnými počítači zbrzdil vývoj neuronových sítí o celá desetiletí.

Předpokládejme učící data ve tvaru $\{(x_i, y_i^t)\}$, kde x_i je vstup neuronu a y_i^t je požadovaná klasifikace. Na začátku učení inicializujeme váhy w na náhodnou hodnotu, nejlépe na hodnoty blízké nule. Poté opakujeme učení (rce. (2.6), [6]) pro všechny hodnoty (x_i, y_i^t) z učící množiny, dokud nedosáhneme požadované přesnosti na množině testovací. Učící krok α můžeme s výhodou v průběhu učení měnit.

$$w = w + \alpha \cdot \left(y_i^t - y_i\right) x_i \tag{2.6}$$

2.3 Učení s učitelem

Učení s učitelem je jednou ze základních úloh klasifikace. Úkolem je roztřídit vstupy do několika množin (např. rozpoznat typ předmětu), přičemž pro učení máme k dispozici učící a validační data - vzorky ze vstupní množiny, u nichž je známa klasifikace.

Oddělení testovací množiny od učící množiny je potřeba kvůli problému přeučení (*overfitting*). Pokud neurony přeučíme, začnou reagovat ne na obecné znaky příslušné množině, ale na konkrétní vzorky z učící množiny. Mimo učící množinu bude mít klasifikace naprosto katastrofální spolehlivost. Použitím jedné sady vzorků pro učení a jiné sady vzorků pro testování neuronové sítě se zabraňuje vzniku tohoto problému.

2.3.1 Backpropagation

Backpropagation (zpětná propagace, překlad je ale řídký), je základní algoritmus pro učení s učitelem. Upravuje váhy neuronů na základě chyby klasifikace (rozdíl očekávané a skutečné hodnoty výstupu neuronové sítě), tak, že tuto chybu posílá na své vlastní vstupy, které jí dále posílají směrem vzad. Základním stavebním kamenem algoritmu backpropagation je fakt, že lze spočítat gradient chybové funkce vůči libovolně zvolené váze (nebo prahu) v neuronové síti s použitím „řetízkového pravidla“ (*chain rule*) o derivaci složené funkce - rovnice (2.7). Je dokázáno, že tento algoritmus konverguje po konečném počtu kroků. [9]

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial p}\frac{\partial p}{\partial w} \tag{2.7}$$

2.4 Učení bez učitele

V tomto případě se neuronová síť snaží poskytovat konzistentní výstup - obdobnou reakci na obdobné podněty. Používá se pro problémy shlukování (*clustering*), snižování dimenze vstupů, detekci anomálií a podobně. [7]

2.4.1 Shlukování

Princip shlukování je velmi jednoduchý a dá se popsat v několika málo bodech. Je potřeba určit „z venčí“ jen počet shluků, do kterých se bude shlukovaná množina dělit. Některé algoritmy si poradí i bez tohoto vstupu, například použitím několika hodnot počtu - je potom na uživateli, aby vybral výsledek, který se zdá nejvíce vhodný.

1. Náhodně umísti reprezentanty shluků (na místo některého z prvků množiny)
2. Vytvoř shluky - každý prvek množiny náleží shluku, jehož reprezentant je v určitém smyslu nejblíže
3. Umísti reprezentanta do středu (např. těžiště) shluku
4. Pokud v důsledku pohybu reprezentanta nějaký prvek množiny změnil příslušnost, opakuj od kroku 2, jinak konec

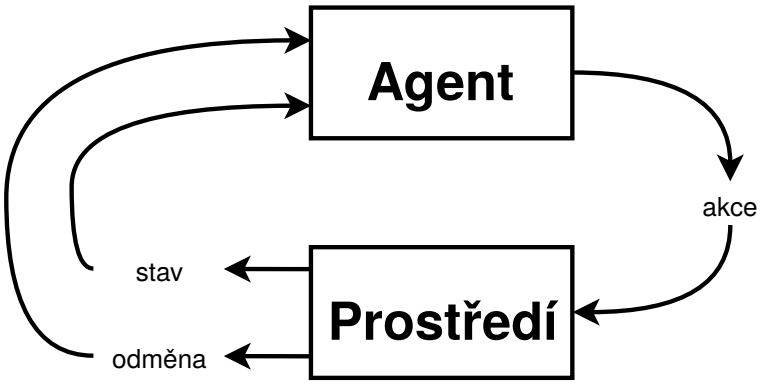
[1] Probíhaly pokusy učit vícevrstvý perceptron za použití evolučních technik.

3 Posilované učení

[10, 11, 13, 26, 28]

Posilované učení (též nazývané zpětnovazební) je druh strojového učení. Spočívá v interakci agenta (učeného systému) s prostředím, jak je schematicky znázorněno na obr. 3.1. Agent je pomocí odměn a trestů motivován k tomu, aby na prostředí reagoval žádoucím způsobem. Na rozdíl od klasického učení s učitelem nejsou potřeba předem připravené vzorky správného chování agenta. Agent sám prozkoumává prostředí prostřednictvím akcí ($a \in A$), v daném stavu agent usuzuje na ohodnocení stavů samotných a postupně se učí dosáhnout cíle procházením stavů s lepším ohodnocením.

3.1 Prvky systému posilovaného učení



Obrázek 3.1: Systém posilovaného učení
podle [13]

3.1.1 Prostředí

Aby mohl agent v daném prostředí fungovat, musí být prostředí pro agenta dostatečně dobře pozorovatelné - ať už reálné prostředí prostřednictvím senzorů robotu nebo simulované prostředí prostřednictvím svých stavových proměnných.

3.1.2 Posilovací funkce

Posilovací funkce R určuje odměnu agentovi r na základě aktuálního stavu a jím vykonané akce. Správně definovaná posilovací funkce je nutnou podmínkou pro korektní učení agenta. I když je možné předkládat systému komplikované posilovací funkce, existují typy funkcí, které dokážou cíl agenta popsat snáze.

Pure delayed reward

Posilovací funkce tohoto typu je daná třemi jednoduchými pravidly:

- pokud je stav konečný a cílový, je odměna +1
- pokud je stav konečný a nežádoucí, je odměna -1
- v ostatních stavech je odměna 0.

Avoidance funkce

Posilovací funkce tohoto typu přikládá každé akci odměnu 0, ledaže by se agent jejím následkem dostal do konečného necílového stavu. V takovém případě je odměna -1. Agent se díky tomu naučí „vyhýbat se porážce“.

Minimum time to goal

Posilovací funkce tohoto typu penalizují každou akci, která bezprostředně nevede k cíli odměnou -1. Pokud akce vede k dosažení cílového stavu, je její ohodnocení 0. Protože se agent snaží maximalizovat odměnu, vedou tyto funkce k nalezení cílového stavu v co nejkratším čase, protože každý krok navíc odměnu snižuje.

3.2 Posilované učení - formální model

3.2.1 Markovovský proces

Téměř všechny úlohy posilovaného učení lze formálně převést na Markovovský rozhodovací proces. Je definován jako uspořádaná dvojice $(\mathcal{S}, \mathcal{P})$, kde $\mathcal{S}$ je množina stavů a $\mathcal{P}$ matice pravděpodobnosti přechodů mezi stavy. Aby byl rozhodovací proces Markovovským, musí splňovat **Markovovskou vlastnost**: budoucí stav procesu $(S_{T+1} \in \mathcal{S})$ je daný pouze současným stavem $(S_T \in \mathcal{S})$ a maticí pravděpodobnosti

$$\mathcal{P}_{ss'} = \begin{pmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{pmatrix} = P\left(S_{T+1} = s' | S_T = s\right) \tag{3.1}$$

Stav procesu nezávisí na jeho minulém průběhu.

3.2.2 Markovovský proces s odměnou

Markovovský proces s odměnou je rozšíření Markovovského procesu o odměňování agenta za dosažení určitého stavu. Formálně je definován jako $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$, kde nově zavedený symbol $\mathcal{R}$ je posilovací funkce a γ je slevový faktor. Odměna za dosažení stavu

$$\mathcal{R}_s = \mathbb{E}\left(R_{T+1} | S_T = s\right) \tag{3.2}$$

je dána střední (váženě průměrnou, značeno $\mathbb{E}$) hodnotou odměny, kterou agent obdrží před příští volbou akce.

Slevový faktor $\gamma \in \langle 0; 1\rangle$ má za úkol snížit hodnotu odměny, kterou může agent v budoucnosti dosáhnout. Využívá se pro snížení odměny v nekonečném cyklu na konečnou hodnotu nebo pro pokyn agentovi, aby zvýhodňoval blízké odměny - například pro aplikace ve finančnictví, kdy okamžitý zisk může být důležitější, než zisk v budoucnosti.

S jeho pomocí můžeme definovat kriteriální funkci

$$G_T = R_{T+1} + \gamma R_{T+2} + \gamma^2 R_{T+3} + \gamma^3 R_{T+4} + ... = \sum_{k=0}^{\infty} \gamma^k R_{T+1+k} \tag{3.3}$$

která určuje celkovou (zlevněnou) odměnu, kterou agent může od časového kroku T získat.

3.2.3 Hodnotící funkce (Value function)

S pomocí kriteriální funkce lze definovat hodnotící funkci, která určuje, na kolik je daný stav s výhodný z hlediska následného zisku odměny

$$\begin{aligned} V(s) &= \mathbb{E}(G_T|S_T = s) \\ &= \mathbb{E}\left(R_{T+1} + \gamma R_{T+2} + \gamma^2 R_{T+3} + \gamma^3 R_{T+4} + ...\right) \\ &= \mathbb{E}\left(R_{T+1} + \gamma\left(R_{T+2} + \gamma R_{T+3} + \gamma^2 R_{T+4} + ...\right)\right) \\ &= \mathbb{E}(R_{T+1} + \gamma G_{T+1}|S_T = s) \\ &= \mathbb{E}(R_{T+1} + \gamma V(S_{T+1})|S_T = s) \end{aligned} \tag{3.4}$$

Rovnice odvozená v 3.4 na posledním řádku se nazývá Bellmanova. Lze ji snadno zapsat maticově

$$\overline{V(S)} = \mathcal{R} + \gamma\mathcal{P}\overline{V(S)} \tag{3.5}$$

ovšem její přímé řešení - se složitostí $N\left(O^3\right)$ - je možné jen pro malé problémy. Pro větší problémy existují řešení iterativní: řešení za pomoci dynamického programování[2], metody Monte-Carlo a Temporal-Difference learning.

3.2.4 Markovovský rozhodovací proces

Pokud do Markovovského procesu s odměnou zařadíme možnost rozhodování, dostáváme Markovovský rozhodovací proces (MRP), který se používá jako model problému pro posilované učení. Jde o pětici $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, kde nový prvek $\mathcal{A}$ je množina akcí, které jsou dostupné agentovi. Ostatní prvky systému - pravděpodobnostní matici a posilovací funkci - je nutné rozšířit tak, aby vliv volby akce agentem obsahovaly

$$\mathcal{P}_{ss'} = P(S_{T+1} = s'|S_T = s, A_t = a) \tag{3.6}$$

$$\mathcal{R}_s = \mathbb{E}(R_{T+1}|S_T = s, A_t = a) \tag{3.7}$$

3.2.5 Výběrová strategie

V MRP lze již definovat výběrovou strategii. Výběrová strategie π určuje agentovi způsob, jakým volit akci z množiny dostupných akcí. Cílem učení je naučit agenta volit dobré akce, čehož může být dosaženo dobrou výběrovou politikou anebo dobrou znalostí Q-funkce (viz 3.2.6).

3.2.6 Q-funkce

Q-funkce (*action-value function, Q function*) určuje agentovi atraktivitu akce v daném stavu

$$Q_\pi(s, a) = \mathbb{E}(G_T|S_T = S) \tag{3.8}$$

Je zřejmé, že agent se v průběhu učení snaží nalézt optimální výběrovou strategii π^*, která v každém možném počátečním stavu povede k maximalizaci jeho odměny (a díky povaze posilovací funkce povede k cílovému stavu a žádoucímu chování). Aby agent mohl optimální strategii určit, potřebuje k tomu dostatečně dobrou aproximaci hodnotící a/nebo Q-funkce.

[2] dostupné jen, známe-li systém plně

3.2.7 Optimální výběrová strategie

Na počátku je Q-funkce neznámá a agent se v systému chová víceméně náhodně. Díky interakci agenta s prostředím se agent učí nové informace o systému a odchylka aproximace postupně klesá. Na konci učení se agent dostane do stavu, kdy už má k dispozici dostatečně dobrou aproximaci funkce a může rozumně hodnotit stavy.

Optimální Q-funkce je daná rovnicí

$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a) \tag{3.9}$$

obdobně optimální hodnotící funkce

$$V^*(s) = \max_{\pi} V_\pi(s) \tag{3.10}$$

Při známé Q funkci lze potom odvodit optimální výběrovou strategii

$$\pi^*(a|s) = \left\{(a,s)\,|\forall s : a = \arg\max_{\pi} Q^*(s,a)\right\} \tag{3.11}$$

Taková strategie ve všech stavech dosahuje nejméně tak dobré hodnoty hodnotící funkce jako všechny ostatní možné strategie. Literatura uvádí (např. [28]), že pro každý MRP existuje alespoň jedna optimální strategie a že existuje-li více optimálních strategií, všechny sdílejí optimální hodnotící a optimální Q-funkci.

4 Algoritmy posilovaného učení - základní přístupy

[11, 13, 15]

Algoritmy posilovaného učení dělíme na

- **modelové** (agent má přístup k modelu prostředí) vs. **bezmodelové**
- **on-policy** (agent při učení používá přímo učenou strategii) vs. **off-policy**
- **epizodické (off-line)** (agent se učí z ucelené epizody, která musí v konečném čase dojít do konečného stavu) vs. **průběžné (online)**

4.1 Monte-Carlo

Zřejmě nejednodušší algoritmus posilovaného učení je algoritmus Monte-Carlo. MC je algoritmus epizodický, bezmodelový, off-policy. MC experimentálně určuje hodnotící funkci jako

$$V(s) = \frac{\sum_{t=1}^{T} \mathbf{1}[S_t = s] G_t}{\sum_{t=1}^{T} \mathbf{1}[S_t = s]} \tag{4.1}$$

kde $\mathbf{1}[S_t = s]$ je binární indikátor, který určuje, jestli se stav v daném časovém kroku započítá (můžeme započítávat všechny průchody stavem nebo např. jen první dosažení). Obsažením akce v indikátoru dostaneme výpočet Q-funkce

$$Q(s,a) = \frac{\sum_{t=1}^{T} \mathbf{1}[S_t = s, A_t = a] G_t}{\sum_{t=1}^{T} \mathbf{1}[S_t = s, A_t = a]} \tag{4.2}$$

Tuto použijeme opakovaně k určení nové strategie $\forall s : \pi(s) = \arg\max_{a \in A} Q(s,a)$. Touto strategií (v jejím ε-hladovém provedení, aby bylo zajištěno prozkoumávání prostředí) dostaneme novou epizodu

$$(s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T, s_T)$$

s níž určíme nový odhad $Q(s,a)$ dle rovnice (4.2). MC konverguje k Q-funkci minimalizující kvadrát odchylky od pozorovaných hodnot. Nepřenáší pozorované zkušenosti z jiných epizod, protože nevyužívá Markovovskou vlastnost procesu. Díky tomu je vhodnější použít v situacích, které Markovovskou vlastnost nesplňují za hranici tolerovatelnosti.

4.2 TD učení (temporal-difference learning)

TD algoritmy jsou online bezmodelové algoritmy. Předpokládejme, že agent vykonáním akce a přejde ze stavu s do stavu s' a za tento přechod obdrží odměnu r. Potom lze online upravit určení ohodnocení stavů podle rovnice (4.3) upravit odhad ohodnocení stavu s na základě ohodnocení stavu s'.

$$V(s) += \alpha(r + \gamma V(s') - V(s)) \tag{4.3}$$

kde α je učící krok, typicky se pohybující okolo 0,05. Takovýto postup nazveme algoritmem TD(0). Člen $r + \gamma V(s')$ se označuje jako *TD-target*, člen $\delta = r + \gamma V(s') - V(s)$ označujeme jako TD chybu (*TD-error*). Rozšířením TD(0) učení o provedení n mezikroků mezi počátečním stavem a stavem, kde použijeme současný odhad ohodnocení dostáváme TD(n) učení

$$V(s) += \alpha\left(\sum_{i=0}^{n-1} \gamma^i r_{i+1} + \gamma^n V(s_n) - V(s)\right) \tag{4.4}$$

Pokud budeme zvětšovat $n \to \infty$, dostáváme MC algoritmus. Omezit se na fixní počet kroků je nevýhodné z hlediska využití znalostí, které získáme ve vyšším časovém horizontu. Spojení výhod MC a TD algoritmů nabízí

algoritmus TD(λ). Toto rozšíření navíc nepřináší další výpočetní náročnost. Dopředná varianta TD(λ) ale ztrácí online učení.

$$G^{\lambda} = (1-\lambda)\sum_{n=0}^{\infty}\lambda^{n}\sum_{i=0}^{n}\gamma^{i}r_{i+1} \tag{4.5}$$

$$V(s) += \alpha\left(G^{\lambda} - V(s)\right) \tag{4.6}$$

Zpětná varianta TD(λ) učení používá „podíly zodpovědnosti“ (*eligibility trace*): každý stav je zodpovědný za celkový výsledek podle toho, jak moc v minulosti se vyskytl a jak často byl navštíven.

$$E_0(s) = 0 \tag{4.7}$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}\left[S_t = s\right]$$

S použitím podílu zodpovědnosti a TD-chyby δ můžeme zformulovat zpětný TD(λ) algoritmus

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{4.8}$$

$$\forall s : V(s) += \alpha\delta_t E_t(s) \tag{4.9}$$

Použití TD bývá výhodnější než použití MC, ačkoliv je více citlivé na počáteční odhad hodnotící funkce. Lze ale použít pro učení v nekonečných procesech.

4.3 Iterativní optimalizace strategie

Doposud představené algoritmy byly schopné strategii pouze ohodnotit, ale nedokázaly strategii formulovat s ohledem na její optimalizaci. Toho dosáhneme s použitím iterativní optimalizace. Tento postup se skládá ze dvou kroků

1. Ohodnocení dané strategie
2. Vylepšení ohodnocené strategie

Strategii můžeme ohodnotit jak MC, tak TD algoritmy, pro vylepšení používáme ε-hladový přístup, abychom zaručili, že se stavový prostor bude dostatečně prozkoumávat.

4.4 SARSA

SARSA (z anglické zkratky state-action-reward-state-action) je algoritmus, který patří do skupiny *temporal-difference learning* algoritmů. Je průběžný, on-policy a bezmodelový.

ε

Význam parametrů použitých v algoritmu 1 je následující:

$\gamma \in (0;1\rangle$ je slevový faktor

ε je parametr „hladovosti“ algoritmu

α je učící krok.

Q-tabulka může být inicializována náhodně, pouze s tím omezením, že pro konečné stavy musí být hodnota nula. Podobně jako prostý TD algoritmus, i SARSA může být rozšířena do podoby SARSA(λ).

Algoritmus 1 SARSA

loop ∞
  S_1 je počáteční stav.
  for t=1,2,... **do**
    Nechť $A_t = \begin{cases} \text{náhodně} & \text{s pravděpodobností } \varepsilon \\ \arg\max_{a \in A} Q(S_t, a) & \text{s pravděpodobností } 1-\varepsilon \end{cases}$
    Vykonáním A_t agent získá odměnu R_{t+1} a informaci o novém stavu S_{t+1}
    Nechť $A_{t+1} = \begin{cases} \text{náhodně} & \text{s pravděpodobností } \varepsilon \\ \arg\max_{a \in A} Q(S_{t+1}, a) & \text{s pravděpodobností } 1-\varepsilon \end{cases}$
    Nechť $Q(S_t, A_t) += \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
    if $S_{t+1} \in S_{konečné}$ **then break**
  end for
end loop

podle [13]

Algoritmus 2 Q-učení

loop ∞
  S_1 je počáteční stav.
  for t=1,2,... **do**
    Nechť $A_t = \begin{cases} \text{náhodně} & \text{s pravděpodobností } \varepsilon \\ \arg\max_{a \in A} Q(S_t, a) & \text{s pravděpodobností } 1-\varepsilon \end{cases}$
    Vykonáním A_t agent získá odměnu R_{t+1} a informaci o novém stavu S_{t+1}
    Nechť $Q(S_t, A_t) += \alpha(R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t))$
    if $S_{t+1} \in S_{konečné}$ **then break**
  end for
end loop

podle [13]

4.5 Q-učení

Zatímco SARSA patří do skupiny on-policy algoritmů (hodnota Q-funkce se upravuje na základě Q-hodnoty stavu získaného strategií $\pi(s) = a$), Q-učení patří mezi off-policy algoritmy (pro úpravu Q-hodnoty stavu se používá jiná strategie, popř. čistě hladový přístup).

4.6 REINFORCE

Explicitní určování Q-hodnot může být náročné paměťově i výpočetně. V mnohých případech, zvláště pokud chceme pracovat se spojitým oborem akcí, může být vhodné neurčovat přímo parametry Q-funkce, ale parametrizovat strategii samotnou. Parametrizovanou strategii můžeme vylepšovat i velmi hrubými metodami se zaručenou konvergencí k alespoň lokálnímu optimu. Jedním z takových přístupů je algoritmus REINFORCE, který je adaptací obyčejného MC algoritmu pro gradientní postup[3].

V algoritmu 3 symbol θ označuje vektor parametrů strategie a ∇_θ je operátor gradientu vzhledem k θ. Je samozřejmě nanejvýš vhodné mít strategii formulovanou tak, aby byla vzhledem k θ diferencovatelná.

[3]Se všemi nevýhodami, které MC přináší - zejména nutnost učit se z ucelených epizod.

Algoritmus 3 REINFORCE

Inicializuj θ náhodně.
loop ∞
    Interakcí s prostředím podle π_θ získej epizodu $(s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T, s_T)$
    for t=0,1,2,...,T-1 **do**
        $\theta += \alpha \nabla_\theta \log \pi_\theta (s_t, a_t) v_t$
    end for
end loop

podle [13]

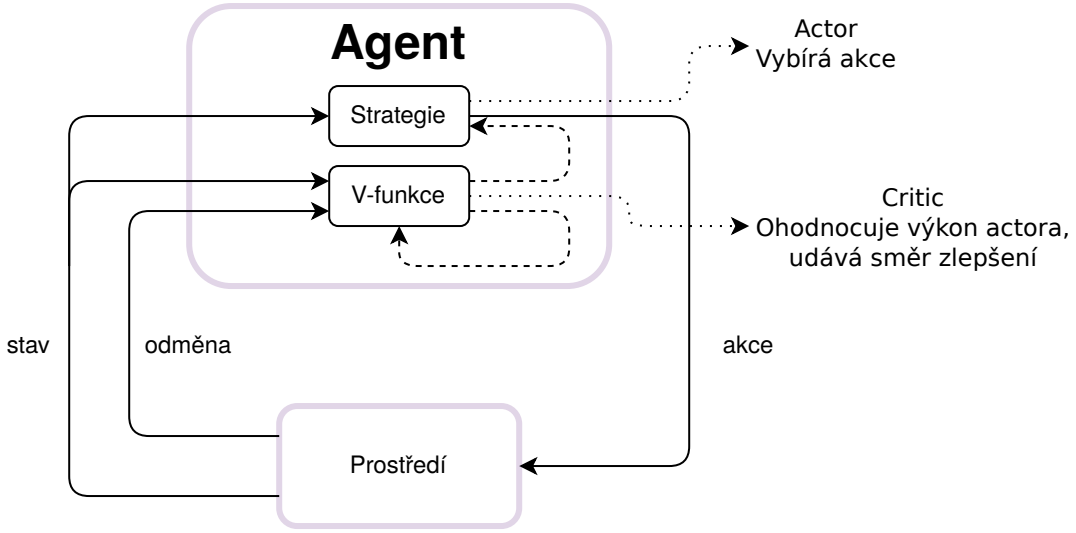
4.7 Actor-critic přístup

Použití prostého REINFORCE algoritmu je na mnohých problémech neefektivní, učení je pomalé a vykazuje vysoký rozptyl „výkonu“ v průběhu učení. Rozptyl je způsobován hlavně proměnným určením v_t. Pokud k odhadu hodnotící funkce použijeme zvláštní sadu parametrů, dostáváme model v literatuře známý jako actor-critic („herec a režisér“). Jejich vzájemný vztah je naznačen na obr. (4.1). V algoritmu 3 nahradíme výpočet parametrů θ vztahem

$$\theta+ = \alpha \nabla_\theta \log \pi_\theta (s, a)\, Q (s, a; \omega) \tag{4.10}$$

Parametry *actora* se budou pohybovat směrem, který *critic* vzhledem ke svým znalostem považuje za vhodné. Použitím aproximace $Q (s, a; \omega) \doteq Q (s, a)$ se samozřejmě připravujeme o přesné určení gradientu, což však není na závadu.

Critic neřeší samotnou optimalizaci strategie π_θ, pouze určuje její ohodnocení. K vyřešení tohoto problému můžeme použít kteroukoliv uvedenou metodu.



Obrázek 4.1: Actor-critic architektura
podle [14]

Rozptyl se dá dále snížit tak, že ve výpočtu nepoužijeme přímo Q-hodnotu dané akce, ale nějakým vhodným způsobem tuto hodnotu upravíme, například tím, že odečteme hodnotu stavu, ve kterém se agent nachází. Tím dostaneme *advantage* funkci, která vyjadřuje, o kolik si agent polepší, pokud ve stavu s zvolí akci a oproti současnému stavu. Přičtením funkce s nulovou střední hodnotou (vůči akcím, neboť akce jsou výstupem *actora*) nezměníme očekávanou odměnu agenta.

$$A (s, a) = Q (s, a) - V (s) = r(s, a) + \gamma V (s') - V (s) \tag{4.11}$$

Tuto funkci můžeme určit například tím, že *critic* se bude učit jak Q-hodnoty, tak ohodnocení stavů s použitím dvojích parametrů. Tento naivní přístup samozřejmě znamená nárůst složitosti učení kvůli vzrůstajícímu počtu parametrů. Lze ukázat, že pro určení *advantage* si lze vystačit s ohodnocením stavů, čímž problém nárůstu parametrů odpadá.

4.8 Deterministický gradient strategie

[16]

V mnohých aplikacích (např. robotice) nelze použít stochastický přístup, protože náhodnost procesu je zanedbatelná. Přirozeným náhradním řešením je zavést umělou náhodnost ovlivňující vykonanou akci. Stále se ale musíme potýkat s problémem vzrůstající náročnosti na dobu učení s tím, jak vzrůstá dimenze problému. Pokud přejdeme k deterministické strategii (jejíž gradient už nezávisí na volených akcích) a použijeme na její určení aproximaci s parametry θ:

$$a_t = \mu\left(s_t; \theta\right) \tag{4.12}$$

dostáváme kriteriální funkci

$$J\left(\theta\right) = \mathbb{E}_{s\sim\rho^{\mu_\theta}}\left(r\left(s, \mu\left(s; \theta\right)\right)\right) \tag{4.13}$$

a její gradient

$$\nabla_\theta J\left(\theta\right) = \mathbb{E}_{s\sim\rho^{\mu_\theta}}\left(\nabla_\theta \mu\left(s; \theta\right) \cdot \nabla_a Q_{\mu_\theta}\left(s, a\right)|_{a=\mu(s;\theta)}\right) \tag{4.14}$$

5 Algoritmy hlubokého posilovaného učení

Původní metody posilovaného učení založené na dynamickém programování používaly k reprezentaci Q-funkce matici o velikosti (počet stavů × počet akcí). Je zřejmé, že takový přístup v případě agentů pohybujících se ve složitém (spojitém) prostředí s velkým (spojitým) počtem akcí neobstojí. Zde potom nastupují algoritmy založené na použití neuronových sítí v roli univerzálního funkčního aproximátoru.

Algoritmus	Modelovost	On/Off-policy	Prostor stavů	Prostor akcí	Operuje s hodnotou
DQN	Bezmodelový	Off-policy	Diskrétní	Spojitý	Q-hodnota
DDPG	Bezmodelový	Off-policy	Spojitý	Spojitý	Q-hodnota
A3C	Bezmodelový	On-policy	Spojitý	Spojitý	Advantage

Tabulka 1: Srovnání algoritmů hlubokého posilovaného učení

5.1 DQN

[27, 29]

Prvním představeným algoritmem je DQN (*Deep Q Network* - hluboká Q síť). Základní myšlenkou je navrhnout a natrénovat hlubokou neuronovou síť, jejímž vstupem je stav systému a výstupem Q hodnoty zvažovaných akcí[4]. Protože výstupem sítě je volba akce agenta, algoritmus si poradí jen s diskrétní množinou akcí, i když množina stavů může být spojitá.

Algoritmus 4 Hluboké Q učení s *experience replay*

procedure DQL
  Inicializuj paměť D o kapacitě N náhodnou interakcí agenta s prostředím
  Inicializuj náhodně váhy Q sítě θ
  for epizoda = 1, M **do**
    Inicializuj posloupnost $s_1 = \{x_1\}$ a proveď preprocesing $\phi_1 = \phi(s_1)$
    Inicializuj $t = 0$
    while epizoda není ukončena **do**
      Nechť $t = t + 1$
      Zvol akci $a_t = \begin{cases} \text{náhodně} & \text{s pravděpodobností } \varepsilon \\ \max_a Q(\phi(s_t), a; \theta) & \text{s pravděpodobností } 1-\varepsilon \end{cases}$
      Proveď akci a_t, zaznamenej odměnu r_t a vstup x_{t+1}
      Nechť $s_{t+1} = s_t, a_t, x_{t+1}$, proveď preprocessing $\phi_{t+1} = \phi(s_{t+1})$
      Do D ulož $(\phi_t, a_t, r_t, \phi_{t+1})$ na místo nejstarší vzpomínky
      Vyber náhodný vzorek vzpomínek $(\phi_j, a_j, r_j, \phi_{j+1})$ z paměti D
      Nechť $y_j = \begin{cases} r_j & \text{pokud } \phi_{j+1} \text{ je konečné} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{pokud není konečné} \end{cases}$
      Trénuj Q s učitelem za použití ϕ_j jako vstupů a y_j jako výstupů
    end while
  end for
end procedure

podle [29]

[4] Taková architektura byla zvolena v [29], jiné práce používají jiné architektury.

Ztrátová funkce sítě má podobu

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta))^2\right] \tag{5.1}$$

$$y_i = \mathbb{E}_{s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})\,|s,a\right] \tag{5.2}$$

kde

$Q(\phi,a;\theta)$ je aproximace Q funkce neuronovou sítí s váhami θ

$\phi(s)$ je předzpracovávající funkce (provádí např. škálování obrázku, převod na stupně šedé atd.)

$\rho(s,a)$ je rozložení pravděpodobnosti mezi sekvencí stavů a akcí agenta

ε je reprezentace prostředí

Vidíme, že na rozdíl od problému učení s učitelem se požadovaný výstup sítě může měnit v závislosti na vahách. Diferencováním rovnice (5.1) podle vah θ_i dostáváme vztah pro učení aproximátoru v podobě lineární kombinace rysů.

$$\nabla_{\theta_i}\Delta L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta)\cdot\nabla_\theta \Delta Q(s,a;\theta)\right] \tag{5.3}$$

Získaný agent je bezmodelový (nemá žádný explicitní model prostředí) a off-policy (při učení se řídí hladovou strategií $a = \max_a Q(s,a;\theta)$, která není totožná s naučenou strategií). Prohledávání je řešeno $1-\varepsilon$ podílem náhodného chování agenta při učení.

5.1.1 Vylepšení základního algoritmu

[30, 31, 32]

Algoritmus 4 ve své základní podobě trpí několika nepříjemnostmi.

Fixace cílové hodnoty (cílová síť)

Podle Bellmanovy rovnice (3.4) algoritmus používá totožné parametry k určení hodnoty Q funkce pro současný stav i pro odhad ohodnocení následujících stavů. Díky tomu se objevuje vysoká korelace mezi váhami sítě a ohodnocením stavů. Tato korelace je nežádoucí, protože při učení Q hodnoty, kdy se síť změnou parametrů učí minimalizovat odchylku mezi skutečnou a výstupní hodnotou změnou vah neustále posouváme odhad skutečné hodnoty Q funkce.

Použitím rozdílných parametrů (označme $\theta^{(1)}$ váhy sítě odhadující a $\theta^{(2)}$ váhy sítě, kterou učíme) tento problém můžeme potlačit: použitím vah $\theta^{(1)}$ k odhadu skutečné hodnoty Q funkce (v rovnici (5.3) ve členu $r + \gamma \max_{a'} Q(s',a';\theta_{i-1})$) a vah $\theta^{(2)}$ pro odhad současné hodnoty snížíme rychlost, s jakou „utíká“ chyba odhadu. Ve vhodných intervalech potom provedeme aktualizaci vah $\theta^{(1)} = \theta^{(2)}$. Aktualizaci vah můžeme provádět i průběžně, použitím váhových parametrů

$$\theta^{(1)} = \tau\cdot\theta^{(1)} + (1-\tau)\,\theta^{(2)} \tag{5.4}$$

Dvojitá DQN

Při určování hodnoty stavu dle rovnice (3.4) tiše předpokládáme, že nejlepší stav má skutečně nejvyšší ohodnocení. Při řešení problému nevadí, pokud jsou všechny stavy nadhodnoceny podobně, ale pokud tomu tak není, síť nemůže řádně konvergovat ([17]). Pokud problém

$$Q(s,a) = r(s,a) + \gamma\left[Q\left(s', \left[\arg\max_{a'} Q(s',a')\right]\right)\right] \tag{5.5}$$

rozdělíme na dvě části (použijeme jednu neuronovou síť ke zjištění $a' = \arg\max_{a'} Q(s',a')$ a druhou k vyhodnocení $Q(s',a')$, významně snížíme vliv nadhodnocení akce, pomůžeme konvergenci a rychlosti učení.

Duální DQN (DDQN)

Tento postup rozděluje problém určení Q hodnoty akce v daném stavu na dva problémy následujícím způsobem:

$$Q(s,a) = [r(s,a)] + [V(s)] \tag{5.6}$$

Použitím dvou sítí k odhadu zvlášť hodnoty stavu a odhadu odměny můžeme docílit toho, že se do hodnoty akce v daném stavu snadno přenese hodnocení samotného stavu. Díky tomu nemusíme v daném stavu tyto akce provádět, pokud volit akci nemá vzhledem k ovlivnění prostředí smysl.

5.2 DDPG

[12, 18]

Aplikací podobného postupu, který byl použit pro zisk DQN na deterministický gradient strategie byl odvozen algoritmus DDPG - hluboký deterministický gradient strategie. Využívá stejné úpravy jako DQN: obsahuje replay buffer a cílovou síť a je postaven na architektuře actor-critic. Algoritmus 5 je pro jednoduchost uveden bez preprocesingu a dalších rušivých částí.

Algoritmus 5 DDPG

procedure DDPG
  Náhodně inicializuj síť critic $Q\left(s,a;\theta^{Q}\right)$ váhami θ^{Q} a actor $\mu\left(s;\theta^{\mu}\right)$ s váhami θ^{μ}
  Inicializuj cílové sítě critic $Q'\left(s,a;\theta'^{Q}\right)$ váhami $\theta^{Q'} = \theta^{Q}$ a actor $\mu'\left(s;\theta^{\mu'}\right)$ s váhami $\theta^{\mu'} = \theta^{\mu}$
  Inicializuj paměť D náhodnou interakcí agenta s prostředím
  loop∞
    Nechť s_1 je počáteční stav
    Nechť N je náhodný (např. Ornsteinův–Uhlenbeckův) proces
    for t=1,2,... **do**
      Nechť akce $a_t = \mu\left(s_t;\theta^{\mu}\right) + N_t$
      Proveď akci a_t, zjisti nový stav s_{t+1} a odměnu r_{t+1}
      Do D ulož přechod $(s_t, a_t, r_t, s_{t+1},)$
      Vyber náhodnou skupinu C přechodů (s_j, a_j, r_j, s_{j+1}) z D
      Nechť $y_j = r_j + \gamma Q'\left(s_{j+1}, \mu'\left(s_{j+1};\theta^{\mu'}\right);\theta^{Q'}\right)$
      Nauč critic minimalizací $\frac{1}{C}\sum_j \left(y_j - Q\left(s_j,a_j;\theta^{Q}\right)\right)^2$
      Nauč actora přechodem po gradientu

$$\nabla_{\theta^{\mu}} J\left(\theta^{\mu}\right) \approx \frac{1}{C}\sum_{j} \nabla_a Q\left(s,a;\theta^{Q}\right)|_{s=s_j,a=\mu(s_j)}\ \nabla_{\theta^{\mu}}\mu\left(s;\theta^{\mu}\right)|_{s=s_j}$$

      Aktualizuj váhy cílových sítí
    end for
  end loop
end procedure

podle [18]

Použitý náhodný proces N je ekvivalentem ε-hladové strategie v případě DQN, zajišťuje agentovi prohledávání stavového prostoru. Díky jeho přítomnosti se DDPG řadí mezi off-policy bezmodelové algoritmy.

Algoritmus si díky použití actor-critic dobře poradí i se spojitým prostorem akcí, neboť actor předkládá své akce k ohodnocení criticovi, který tak má k ohodnocení jen spočetně mnoho akcí.

5.2.1 MADDPG

MADDPG (Multi Agent DDPG) je rozšíření základního algoritmu DDPG na prostředí Markovovských her - interakce více agentů v jednom prostředí. V tomto modelu se agenti pohybují ve společném prostředí, ale každému je příslušná jeho vlastní množina akcí a pozorování okolí. Změna stavu prostředí závisí na akcích všech agentů, ale každý agent si drží vlastní strategii, ve které zohledňuje jen vlastní akce a pozorování.

Kritik se učí aproximovat Q-funkci

$$Q_i^{\mu^\theta}\left(s, a_1, a_2, \ldots, a_n\right) \tag{5.7}$$

kde $i \in \{1, 2, \ldots, n\}$ je index agenta. Každá složka Q-funkce příslušná agentovi je učena zvlášť, což agentům umožňuje mít různé (třeba i protichůdné) cíle. Každému agentovi přísluší vlastní síť actor, což agentům zajišťuje vzájemnou nezávislost.

Rovnice gradientu actor sítě, resp. ztrátové funkce critica se změní na

$$\nabla_{\theta_i} J\left(\theta_i\right) = \mathbb{E}_{s,a\sim D}\left(\nabla_{a_i} Q_i^{\mu^\theta}\left(s, a_1, \ldots, a_n\right) \nabla_{\theta_i} \mu^{\theta_i}\left(s\right)|_{a_i=\mu^{\theta_i}(s)}\right) \tag{5.8}$$

$$L\left(\theta_i\right) = \mathbb{E}_{s,\{a_i\},\{r_i\},s'}\left[\left(Q_i^{\mu^\theta}\left(s, a_1, \ldots, a_n\right) - \left(r_i + \gamma Q_i^{\mu'^\theta}\left(s', a'_1, \ldots, a'_n\right)|_{a'_j=\mu'^{\theta_j}}\right)\right)^2\right] \tag{5.9}$$

kde ${\mu'}_i^\theta$ je i-tá aktualizovaná cílová strategie.

MADDPG zavádí nový pojem *seskupení strategií*, jehož zavedení má za úkol snížit rozptyl vzniklý interakcí více agentů s podobnými strategiemi. Každý agent se učí několik strategií, přičemž na začátku učící epizody si jednu z nich náhodně zvolí. Úprava všech agentových strategií probíhá najednou, ale pro každou strategii jsou z paměti brány vzpomínky náhodně (i ze vzpomínek jiných agentů).

MADDPG se snadno dá rozšířit do podoby částečně pozorovatelných Markovovských her, kdy agenti nemají přístup k celé reprezentaci stavu, ale jen k vlastnímu pozorování (viz např. [19]). Algoritmus dokáže dobře konvergovat i při použití odhadu strategií ostatních agentů.

5.2.2 D4PG

Algoritmus D4PG (Distributed Distributional Deterministic Policy Gradients) staví na algoritmu DDPG a rozšiřuje ho v několika bodech

Distributional Critic

Critic síť odhaduje Q-hodnotu jako střední hodnotu náhodné veličiny příslušící rozdělení pravděpodobnosti $Z_\omega : Q_\omega\left(s, a\right) = \mathbb{E}\left(Z_\omega\left(s, a\right)\right)$.

N-krokové odměny

Obdobně jako v případě TD(N) učení, algoritmus pracuje s určením TD-cíle pomocí N následujících kroků (viz rovnici (4.4)). Aby takové chování dávalo smysl, ze vzpomínkové paměti se musí pro učení vybírat N po sobě jdoucích kroků jediného agenta.

Paralelizace agentů

Algoritmus používá více agentů pracujích paralelně v oddělených prostředích, zatímco sdílejí jedinou paměť vzpomínek D.

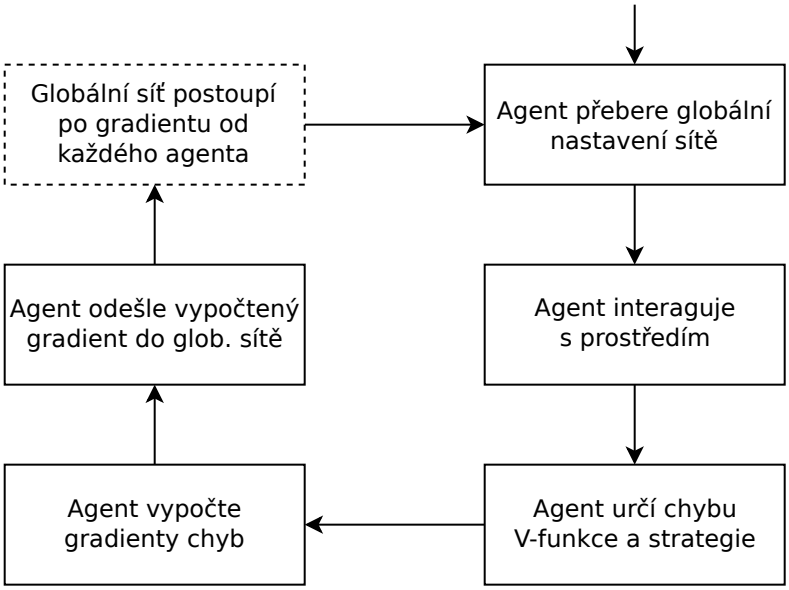
Prioritizace výběru vzpomínek

Vzpomínky uchovávané v paměti se vybírají s nerovnoměrnou pravděpodobností. Původní rovnoměrné rozdělení pravděpodobnosti znamenalo, že vzpomínky byly vybírány z paměti s pravděpodobností odpovídající jejich četnosti. Prioritizací vzpomínek podle jejich významu se dosahuje lepšího výkonu sítě ([21]).

5.3 A3C

A3C je algoritmus patřící mezi bezmodelové on-policy algoritmy. Je od počátku stavěn jako asynchronní, takže se dobře škáluje při použití více výpočetních jednotek. Původní verze algoritmu používá dopřednou variantu výpočtu, pracuje tedy off-line.

Algoritmus používá více paralelně běžících agentů v oddělených prostředích. Každý agent asynchronně posílá své příspěvky globální síti na konci epizody a na začátku epizody se synchronizuje s globální sítí, jak je naznačeno na obr. 5.1.



Obrázek 5.1: Schema práce jednoho agenta v algoritmu A3C
Podle [22]

5.3.1 A2C

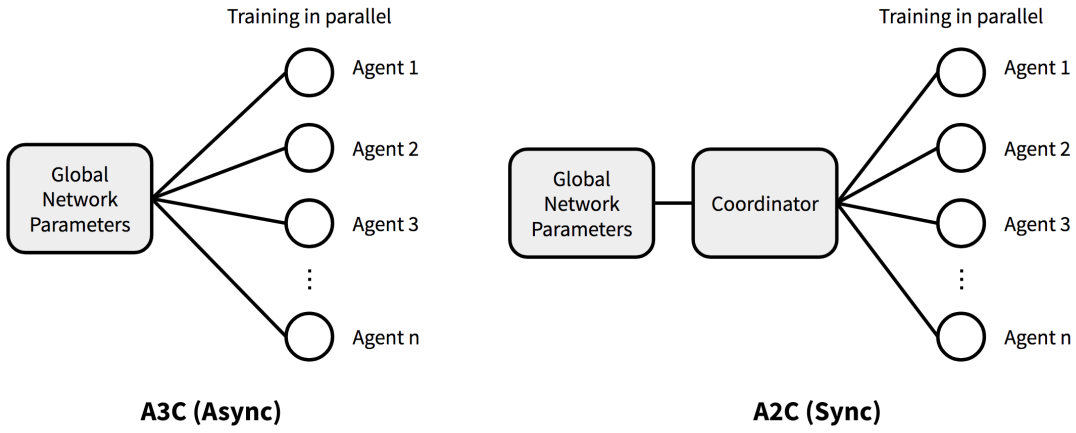
A2C je synchronní/synchronizovaná varianta A3C. Podle provedení může např. nechat všechny agenty dokončit interakci s prostředím a zaslat hlavní síti svůj vypočtený gradient, než dovolí agentům zahájit novou epizodu (jak je znázorněno na obr. 5.2), případně pracuje jen s dvojicí sítí, kdy se první síť použije k interakci agenta s prostředím a druhá pro samotné učení. I v této variantě může vedle sebe koexistovat velké množství různých prostředí ([24]).

Algoritmus 6 A3C

Require: globální parametry θ, θ_v

procedure A3C
  Nechť hodiny agenta $t = 0$
  loop ∞
    Resetuj gradienty $d\theta = 0, d\theta_v = 0$
    Synchronizuj parametry s globálními $\theta' = \theta, \theta'_v = \theta_v$
    Nechť $t_s = t$
    Nechť s_t je počáteční stav
    repeat
      Nechť $a_t = \pi\left(a_t | s_t; \theta'\right)$
      Proveď a_t, zaznamenej odměnu r_t a nový stav s_{t+1}
      Tikni hodinami $t+ = 1$
    until $s \in S_{konečné}$ OR $t - t_s == t_{max}$
    Nechť $R = \begin{cases} 0 & \text{pokud je } s_t \text{ konečný} \\ V\left(s_t; \theta'_v\right) & \text{pokud } s_t \text{ není konečný} \end{cases}$
    for $i = t-1, t-2, ..., t_s$ **do**
      Nechť $R = r_i + \gamma R$
      Nechť $d\theta+ = \nabla_{\theta'} \log \pi\left(a_i | s_i; \theta'\right)\left(R - V\left(s_i; \theta'_v\right)\right)$
      Nechť $d\theta_v+ = \frac{\partial\left(R - V\left(s_i; \theta'_v\right)\right)}{\partial \theta'_v}$
    end for
    Asynchronně pošli $d\theta$ a $d\theta_v$ do globální sítě k postupu
  end loop
end procedure

přebráno z [23]

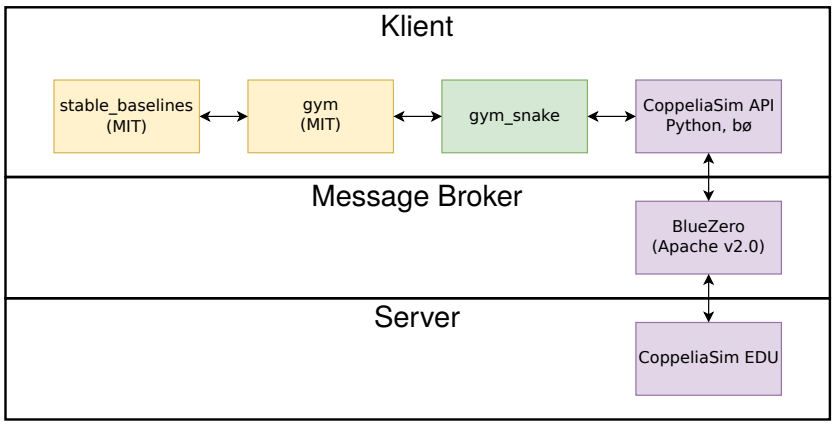


Obrázek 5.2: Rozdíl architektur v algoritmech A3C a A2C
převzato z [12]

6 Návrh modelu robotu

6.1 Volba prostředků simulace

Autor pro vytvoření simulace zvolil konvenční prostředky - vlastní simulaci provádí simulátor CoppeliaSim (nástupce V-Repu od stejného výrobce), řízení robotu neuronovou sítí a jeho učení probíhá v odděleném procesu. Programovacím jazykem je Python, neuronové sítě jsou řešeny knihovnou tensorflow. Propojení mezi klientem (Python) a serverem (CoppeliaSim) zajišťuje message broker BlueZero. Celá architektura je znázorněna na obrázku 6.1. Oddělení procesu učení a simulace umožňuje, aby v případě potřeby mohly být pro obě součásti použity oddělené stroje, např. z důvodu použití rozdílných operačních systémů nebo zvýšení výkonu. Z hlediska latence je vhodné, aby message broker používal stejný stroj jako simulátor, protože při použití grafické karty na učení neuronové sítě je nejnáročnějším procesem samotný simulátor.



Obrázek 6.1: Architektura simulačního experimentu

6.1.1 Volba simulačního prostředí

Na trhu existuje několik druhů software, zabývající se simulací robotiky. Z nich jsou asi nejčastěji používány tyto simulátory:

- Gazebo
- SimSpark
- V-Rep/CoppeliaSim
- Webots

V této práci je použit simulátor CoppeliaSim. Je v oblasti široce používaný, existuje pro něj velké množství návodů a i v neplacené - školní - verzi je mu poskytována podpora od výrobce. Základ kódu je open-source popř. source-available, což poskytuje alespoň jistou auditovatelnost a nezávislost na výrobci.

Simulátor Gazebo poskytuje API pouze pro jazyk C++, jenž je pro postupný vývoj programu typu „úloha posilovaného učení“ méně vhodný. Simulátor Webots byl omezen z hlediska importu složitých tvarů (segment robotu) na formáty, jejichž export z použitého modelovacího software by byl zbytečně komplexní - vyžadoval by použití doplňujícího programu.

6.1.2 Volba programovacího jazyka

Na základě srovnání v tabulce 2 byl zvolen jazyk Python. Je interpretovaný (tedy vhodný pro prototypování), v oblasti široce používaný (s čímž souvisí dostupnost knihoven a dokumentace). Interpretované jazyky jsou ale ze

své podstaty pomalejší než jazyky kompilované přímo do nativního kódu. Jak se ale později ukázalo, úzké hrdlo výkonu je samotný simulátor. I tak by ale bylo možné před kompletním přepisem hotového programu některý z nástrojů určený pro zrychlení běhu programů v Pythonu - např. Numba nebo Cython[5]. Použité rozhraní je b0-API, jelikož původní verze API je výrobcem označena za zastaralou a její použití vykazuje jistou těžkopádnost.

Simulátor zahrnuje potřebné soubory pro použití API v jazycích C/C++, Java, Lua, Matlab/Octave a Python. Pokud je pro programování použito nové rozhraní b0 API, je možné doprogramovat rozhraní v dalších jazycích. Vzhledem k tomu, že rozhraní poskytnutá v distribuci programu jsou dostačující, nebylo této možnosti využito, jakkoliv by použití jazyků R, Julia, Go nebo LISP bylo z programátorského hlediska zajímavé.

API je použito v synchronním režimu, t.j. simulátor sám od sebe nic nedělá a simulační krok se děje jen na pokyn z klienta - díky tomu přepínání procesů simulátor-broker-klient v rámci operačního systému nevyžaduje absolutní načasování.

Jazyk	Běhové prostředí	Typování	Knihovny pro		Licenční politika
			NN	RL	
C/C++	kompilovaný	statické	+	+	dle distribuce (F/OSS dostupné)
Java	interpretovaný	statické	+	+	F/OSS
Lua	interpretovaný	dynamické	+	?	F/OSS
Matlab	interpretovaný	dynamické	+	+	proprietární
Octave	interpretovaný	dynamické	+	?	F/OSS
Python	interpretovaný	dynamické	+	+	F/OSS

Tabulka 2: Srovnání jazyků, pro které je poskytováno rozhraní

Model robotu byl vytvořen volně podle existujícího robotu v majetku UAI. Předloha obsahuje segment hlavy a šest segmentů spojených servomotory. Každý segment obsahuje dvě pasivní kola, která simulují anizotropii břišních šupin hadů: kladou malý odpor při posunu vpřed (valení), ale velký odpor při pohybu v bočním směru (smýkání).

6.2 Tvorba modelu robotu

Nejprve byl celý model vytvořen v programu Autodesk Inventor. Součástí modelu byly vazby mezi segmenty pomocí kolíků i osazení kol na čepy. Tento model však narazil na potíže při dělení a umisťování kloubů v simulátoru.

Proto byl pro experiment použit model částečně vytvořený přímo v simulátoru. V modelovacím software bylo vytvořeno pouze zjednodušené tělo segmentu (obrázek 6.2). Model segmentu je potřeba exportovat do formátu, který může simulační software načíst. Pro přenos byl použit formát *obj*. Tento formát ale neukládá rozměry absolutně, ale jen jejich relativní poměry. To zapříčinilo nutnost provést při importu modelu škálování 1:1000 směrem dolů, neboť modelovací software používá pro export základní rozměr „1 jednotka $\sim$ 1 mm“, kdežto simulátor importuje soubory *obj* v režimu „1 jednotka $\sim$ 1m“.

Tyto segmenty byly poté propojeny klouby typu rotační vazba (jeden stupeň volnosti). Tyto klouby byly v programu nastaveny jako aktivní (ovládatelné) a bylo jim nastaveno uzamčení při nulové rychlosti. Kola byla vytvořena jako primitivní tvar v simulačním software a obdobně zavazbena k tělu segmentu. Jejich kloub byl ale označen jako pasivní, aby byla umožněna volná rotace kola kolem „čepu“. Prvky spojující kola a segmenty byly v tomto modelu zanedbány, podobně jako hlava hada: místo modelu hlavy je použito tělo segmentu. Vzhledem k rozměrům a vzdálenostem použitých překážek není toto zanedbání na závadu.

[5] viz např. články Pavla Tišnovského na serveru root.cz:
Numba: https://www.root.cz/clanky/projekt-numba-aneb-dalsi-pristup-k-prekladu-pythonu-do-nativniho-kodu/
Cython: https://www.root.cz/clanky/rpython-vs-cython-aneb-dvoji-pristup-k-prekladu-pythonu-do-nativniho-kodu/#k10

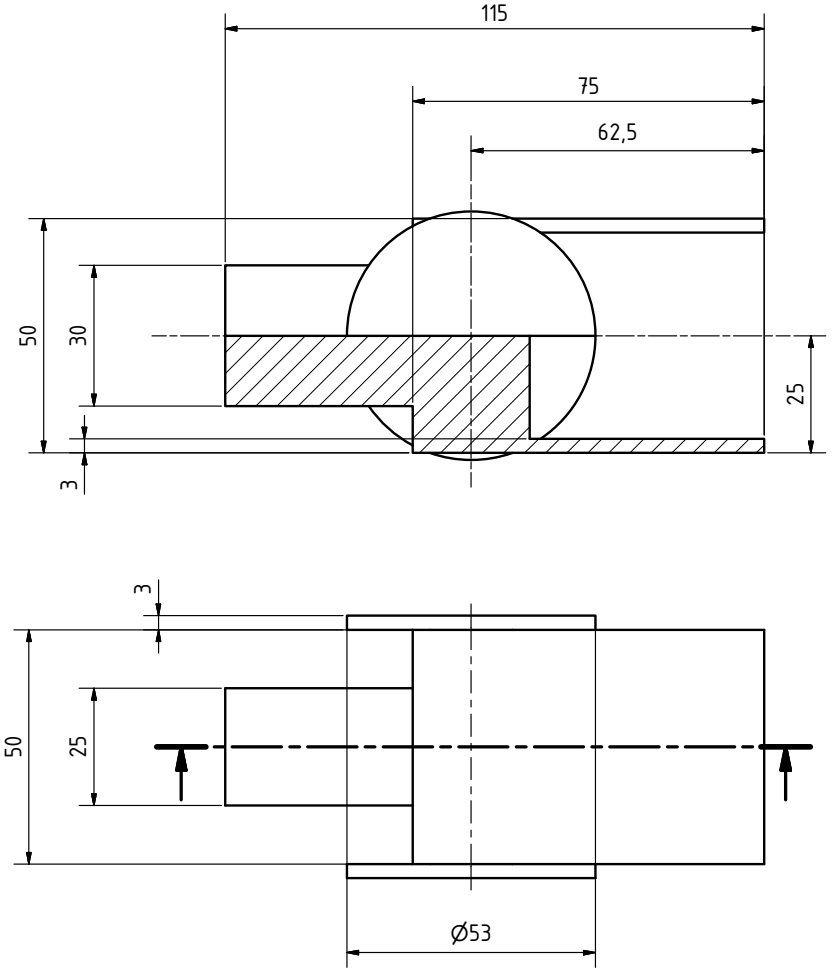
Model robotu byl osazen na hlavě třemi senzory přiblížení pro detekci překážek. Při jejich umisťování je třeba dbát na to, aby detekční kužel neprotínal podlahu. Simulátor sice umožňuje softwarově zabránit senzorům detekování určitého objektu, použití softwarového bloku by ale snížilo použitelnost naučené sítě v praxi.

Při tvorbě modelu je vhodné jednotlivé bloky, ze kterých se model skládá pojmenovávat dostatečně popisně, neboť jméno bloku je zároveň jeho „adresou“, přes kterou pracujeme z externích programů. V modelech byla zvolena konvence dle tabulky 3.

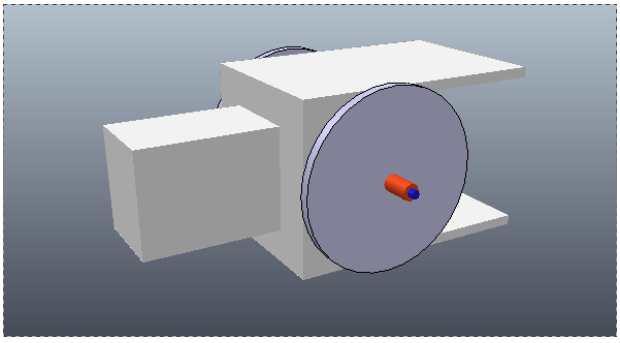
Jméno	Označuje...
elem$i$	i-tý segment těla ($i = 0..6$)
elem$i$_{R, L}	kloub pravého a levého kola i-tého segmentu
w_elem$i$_{R, L}	kolo (wheel) na kloubu elem$i$_{R, L}
joint_ij	kloub spojující i-tý segment s j-tým segmentem
proximity_{left, middle, right}	senzor přiblížení (levý, prostřední, pravý)

Tabulka 3: Konvence pojmenování prvků modelu

Výsledný model je ukázán na obrázku 6.3.

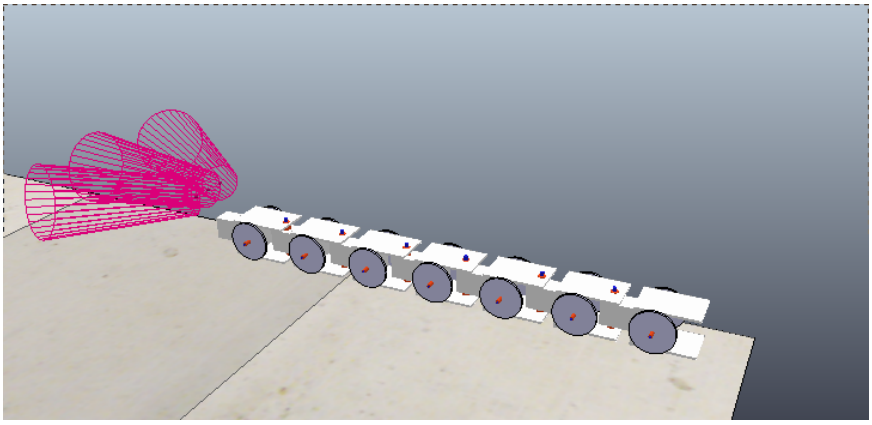


(a) Výkres segmentu



(b) Pohled na segment

Obrázek 6.2: Segment těla robotu



Obrázek 6.3: Pohled na model

7 Simulační experiment

7.1 Východisko

Pro návrh učícího prostředí se nejprve podívejme na to, jak se chová biologická předloha robotu. U hadů se rozeznává několik druhů pohybů [33, 34]:

- lineární pohyb vpřed (*rectilinear locomotion*) - „housenkovitý“ pohyb
- pohyb vlněním (*simple undulation*)
  - v přímém směru (*lateral undulation*, *serpentine locomotion*)
  - v bočním směru (*sidewinding*)
- harmonikový pohyb (*concertina locomotion*)
- přehazování „ze strany na stranu“ (*slide pushing*)

Had, který se pohybuje vlněním v přímém směru postupně vlní celé tělo (vlny počínají u hlavy a pohybují se směrem k ocasu) tak, že střídavě klade záhyby na obě boční strany, jak vidíme na obrázku 7.1a. Směr pohybu je určen hlavou a krkem, zbytek těla pouze dodává potřebné opěrné body. Díky tomu, že had klade část svého těla na obě strany do tvaru opakovaného písmene S, pravo-levé složky sil se navzájem vyruší a výsledkem je pohyb vpřed. Aby tento model pohybu fungoval, je zapotřebí dostatečného tření mezi tělem hada a povrchem, na kterém se had pohybuje. Výhodou je členitý povrch s množstvím drobných překážek, jak ukazuje příklad užovky proužkované na obrázku 7.1b.

[35]

(a) Schematický nákres

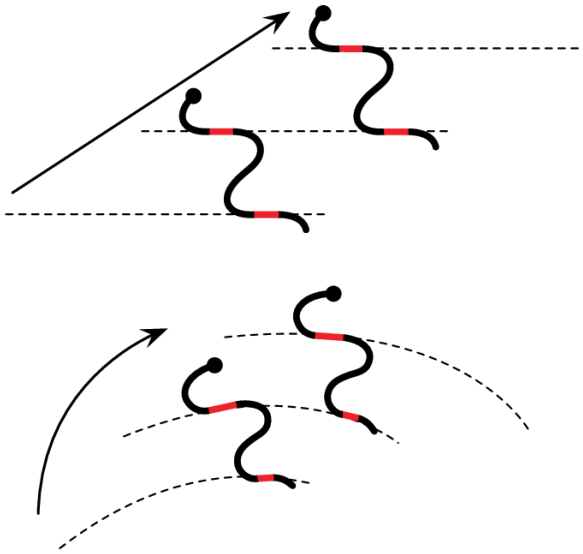
[36]

(b) Užovka proužkovaná ve členitém terénu

Obrázek 7.1: Vlnitý pohyb hada v přímém směru

Pohyb vlněním v bočním směru je typický pro pouštní hady, kteří překonávají jemný a sypký písek. Had se dotýká povrchu jen v částech těla (na obr. 7.2a znázorněno červenou barvou). Díky kombinaci vlnění a přesunu kontaktu s povrchem se had pohybuje bokem vůči jeho „čelnímu“ směru - stopy hada jsou ukázány na obrázku

7.2b. Protože se části těla dotýkající se povrchu rychle střídají, minimalizuje se navíc tepelná výměna s povrchem, díky čemuž mohou pouštní hadi cestovat na horkém písku pohodlně - alespoň relativně vzato.



[37]

(a) Schematický nákres

[38]

(b) Stopy hada v písku

Obrázek 7.2: Pohyb vlněním v bočním směru

Na základě toho bylo výchozí prostředí simulace upraveno

- zvýšením koeficientu tření podlahy v případě problému *undulatory locomotion* na f=100
- snížením koeficientu tření podlahy v případě problému *sidewinding* na f=0.01

7.2 Neuronová síť

Neuronová síť obsahuje čtyři plně propojené vrstvy s aktivační funkcí tanh.

Vstupní vrstva

Do neuronové sítě vstupuje stav agenta (robotu), který byl autorem navržen takto:

1. pozice hlavy v x-ovém směru v globálním souřadném systému
2. pozice hlavy v y-ovém směru v globálním souřadném systému
3. natočení hlavy v z-ové ose v globálním souřadném systému
4. vzdálenost hlavy k cíli v x-ovém směru v globálním souřadném systému
5. vzdálenost hlavy k cíli v y-ovém směru v globálním souřadném systému
6. levý senzor přiblížení - vzdálenost k překážce nebo -1
7. prostřední senzor přiblížení - vzdálenost k překážce nebo -1
8. pravý senzor přiblížení - vzdálenost k překážce nebo -1
9. poloha kloubu „joint_01“ (natočení) v lokálním souřadném systému (v rad)
10. poloha kloubu „joint_12“
11. poloha kloubu „joint_23“
12. poloha kloubu „joint_34“
13. poloha kloubu „joint_45“
14. poloha kloubu „joint_56“
15. rychlost kloubu „joint_01“ (úhlová rychlost v rad/s)
16. rychlost kloubu „joint_12“
17. rychlost kloubu „joint_23“
18. rychlost kloubu „joint_34“
19. rychlost kloubu „joint_45“
20. rychlost kloubu „joint_56“

Skryté vrstvy

Neuronová síť obsahuje dvě skryté vrstvy s 64 neurony v každé z nich. Toto nastavení je provedeno importem MlpPolicy z balíku stable_baselines. Autor přiznává, že jeho zkušenosti v návrhu neuronových sítí jsou nanejvýš omezené a že je pravděpodobné, že toto nastavení nebylo zvoleno optimálně. Na základě konzultace s vedoucím práce je ale toho názoru, že příliš „široká“ neuronová síť je lepší variantou než „úzká“ síť, protože příliš úzká síť s sebou nese riziko nedostatečné komplexity vůči zadané úloze.

Výstupní vrstva

Vrstva obsahuje 6 výstupních neuronů, které udávají požadovanou rychlost tělních kloubů hada („joint_01“ až „joint_56“). Řízení rychlosti vnáší do problému komplexitu navíc - je potřeba neustále hlídat polohu kloubu, řešit rozjíždění a zastavení kloubu v určité poloze „ve vlastní režii“. Řízení polohy by ovšem vyžadovalo dobrou znalost servomotoru v předloze modelu, tudíž vzhledem k době, v níž tato práce vznikala, je autor toho názoru, že i přes nárůst komplexity bylo zvoleno méně špatné řešení.

7.3 Posilovací funkce (odměna)

Posilovací funkce je záměrně konstruovaná pokud možno jednoduše i za cenu nedokonale vystihnutých nároků na robot. Odměna je daná záporně vzatou vzdáleností hlavy k cíli. Pokud ovšem robot opustí vymezený prostor (a tudíž padá), je mu udělena vysoká penalizace.

Tato posilovací funkce neřeší např. celkovou spotřebu energie robotu při cestě, minimalizaci času stráveného cestou řeší pouze implicitně (všechny odměny jsou záporné). Autor si je nedostatků vědom a takovou posilovací funkci zvolil z důvodu vyrovnání komplexity těch nastavení, které vzhledem k nezkušenosti autora bylo možné volit i při konzultaci s literaturou jen odhadem.

7.4 Učící algoritmus

Protože problém je řešen ve spojitém prostoru stavů i akcí, je nutné zvolit učící algoritmus typu actor-critic. Autor zvolil algoritmus A2C z balíku stable_baselines v realizaci s jedním agentem. Algoritmus byl zvolen na základě těchto kritérií:

- přítomnost algoritmu v používané a kvalitní knihovně
- algoritmus je dostatečně nový na to, aby obsahoval nové poznatky z oboru, ale dost starý na to, aby se dal označit za „prověřený časem“
- literatura jej uvádí jako výkonný a robustní (např. [39])

Jedna učící dávka obsahuje 10 000 000 kroků, přičemž program automaticky ukládá stav sítě po každých 1000 krocích, aby bylo možné navázat v případě neplánovaného ukončení programu s co nejčerstvějšími hodnotami. Protože k realizaci neuronové sítě je použita knihovna tensorflow, lze použít standardní logovací rozhraní pro tuto knihovnu (tensorboard) k pohledu přímo dovnitř do učícího procesu. Autor v programu z tohoto důvodu neuvaluje další kritéria na ukončení učení - ve chvíli, kdy bude síť považována za dostatečně naučenou, je možné program přerušit a použít nejčerstvější zálohu jako výstup programu.

Algoritmus byl použit bez dalších úprav nastavení, tedy zejména nastavení hyperparametrů. Autor je toho názoru, že vzhledem k jeho nedostatečným zkušenostem v oblasti posilovaného učení je vhodnější ponechat nastavení hyperparametrů ve výchozím nastavení, neboť případné zásahy by byly spíše na škodu než ku prospěchu.

7.5 Propojení simulačního software s učícím algoritmem

Učící algoritmus je potřeba spustit v určitém prostředí: toto prostředí je scéna simulátoru. Učící algoritmus využívá rozhraní openai.gym. V rámci práce bylo vytvořeno prostředí gym_snake, které implementuje rozhraní openai.gym a napojuje učící algoritmus na model robotu a scénu, ve které se pohybuje. Pro korektní funkčnost algoritmu musí prostředí poskytovat informace o prostoru pozorování a akcí (dimenze prostoru s volitelným omezením na dostupné hodnoty - např. omezení ve výkonu motoru), volitelně též o prostoru možných odměn (s výchozími hodnotami $(-\infty; \infty)$). Dále musí poskytovat metody s následujícími signaturami:

- step(action: self.action_space) → observation: self.observation_space, reward: float, is_done: bool, additional: dict
- reset() → observation: self.observation_space

- close() → ret: int = 1
- render(mode="human": str) → None
- seed(seed=None: Union[None, float]) → seed: float

Prostor pozorování (observation space)

Dimenze prostoru odpovídá rozměru vstupní vrstvy neuronové sítě. Obsahuje omezení na rozměry dostupné „podlahy", natočení kloubů a jejich rychlosti.

Prostor akcí (action space)

Dimenze prostoru odpovídá rozměru výstupní vrstvy. Obsahuje omezení na maximální rychlost kloubu (v obou směrech).

Metoda kroku (step)

Metoda akceptuje jeden parametr - akci agenta - a vrací čtveřici hodnot (stav, odměna, konec simulace, slovník dalších parametrů). Akci odesílá simulátoru prostřednictvím *publishera* b0. Následně spouští synchronní krok simulace a čeká na jeho dokončení. Po dokončení synchronního kroku vyčítá hodnoty stavu prostřednictvím zaregistrovaných callbacků - *subscriberů*.

Autor upozorňuje na nutnost vytvoření subscriberů v samostatném bloku kódu (z tohoto důvodu je lambda callbacku návratová hodnota funkce) - uzávěry v Pythonu sdílí prostor pro uzavřenou proměnnou. Pokud by tedy byly registrovány všechny lambda výrazy ve stejném bloku, callbacky by sdílely hodnotu uzavřené proměnné rovnou poslední hodnotě iterátoru.

Subscribery registrované pro hodnoty stavových proměnných jsou omezeny na jedinou hodnotu ve „frontě", stejně tak subscriber pro reakci na konec synchronního kroku. Tím je docíleno stavu, kdy všechny „výsledky" subscriberů budou obsahovat poslední známou hodnotu stavové proměnné. Tento přístup šetří paměť počítače: jakákoliv jiná než poslední hodnota pro program stejně nemá význam.

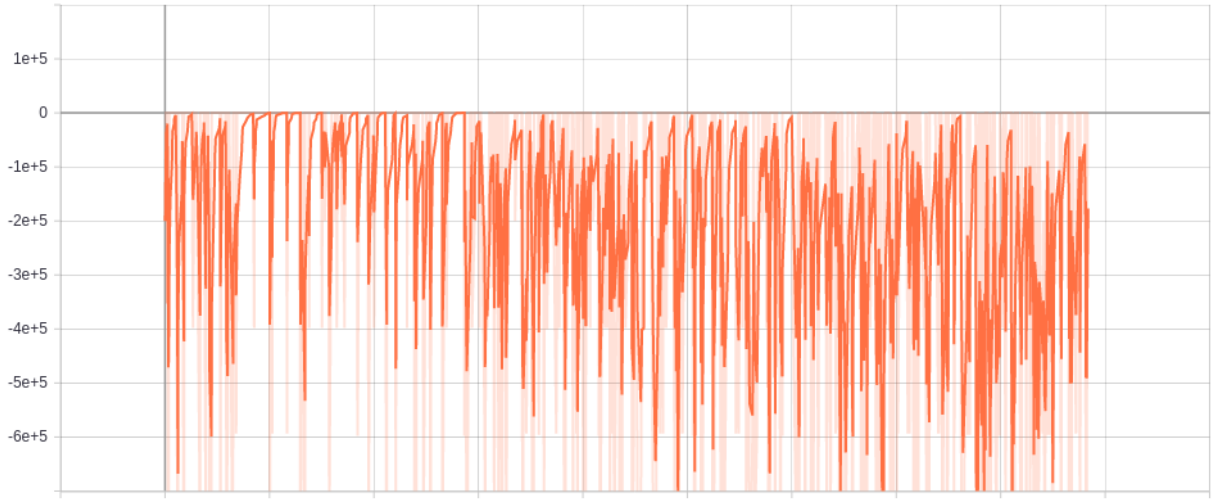
8 Vyhodnocení experimentu

Experiment samotný bohužel narazil na jeho velkou časovou náročnost. I přes použití knihoven CUDA, umožňující paralelizaci výpočtů na GPU nedošlo ve vymezeném čase k jakémukoliv pokroku. Autor identifikoval několik možných příčin tohoto neúspěchu.

- Počáteční poloha robotu je příliš vzdálená jeho cíli. Mohlo by být výhodnější vzdálenost robotu k cíli postupně zvyšovat, aby zejména v počáteční fázi učení cíl mohl být dosažitelný „náhodou“. Tato úprava by mohla zvýšit podíl „pozitivních“ akcí, které tak mohly nastartovat pozitivní cyklus využívání znalostí.
- Časté pády robotu mimo vymezenou oblast způsobily časté resetování simulace. Autor si toto uvědomoval už ve fázi návrhu simulace, ale byl toho názoru, že přirozený vývoj pomůže tuto situaci vyřešit v krátkém časovém horizontu. Při pohledu zpět se jeví jako lepší možnost výrazné zvětšení dovoleného prostoru a přizpůsobení posilovací funkce - pokud by robot opustil „malou“ vymezenou plochu, došlo by k dalšímu snížení hodnoty odměny.
- Trest agenta za vypadnutí z jemu určenému prostoru může být příliš velký. Zpětná propagace této odměny postihuje všechny akce agenta, které jí předcházely, tedy i ty, které mohly být vůči cíli agenta vhodné. V dlouhodobém výhledu sice dojde k tomu, že skutečně špatné akce budou penalizovány více, ale penalizace „správných“ akcí v krátkodobém pohledu přeci jen narušuje proces učení.
- Práce se senzory na vstupu neuronové sítě může být vzhledem k aktivační funkci neelegantní - největší míra zásahu je potřeba v případě, že je překážka nejblíž. Na jednu stranu - směrem do vyšších kladných hodnot potřeba zásahu přirozeně klesá, na druhou stranu je stav „žádná překážka detekována“ indikován hodnotou -1. Monotónně rostoucí hyperbolický tangens nemá jak vyjádřit pokles naléhavosti zásahu na obě strany. Výhodnější by mohlo být použití aktivační funkce ReLU s převrácenou hodnotou vstupu.
- Spojení mezi klientem a serverem v jednom případě - z nedostatku lepšího výrazu - „vyšumělo do ztracena“. Pravděpodobně z důvodu neaktivity spojení mezi klientem a message brokerem bylo toto přerušeno a bylo nutné klient restartovat. Sice bylo použito poslední zálohy neuronové sítě, takže nedošlo ke ztrátě „znalostí“, byla však ztracena „paměť“ algoritmu - úložiště volených akcí a jim odpovídajících odměn, takže bylo nutno tyto vzpomínky nejprve znovu nasbírat, než se ze strany učícího algoritmu mohlo přikročit k jejich používání. Dále došlo v průběhu výpočtu k několika restartům počítače z různých důvodů, což mělo stejné následky.
- K neúspěchu přispěla i volba řízení rychlosti kloubu v těle robotu, třebaže tato volba byla z pohledu autora odůvodněna (viz 7.2 na straně 37).

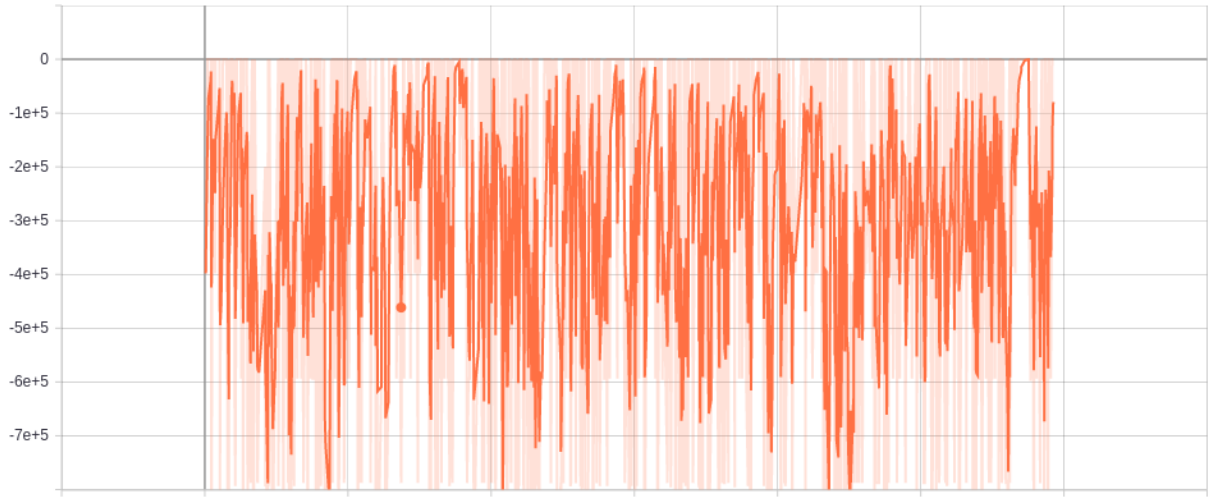
Na grafech 8.1 až 8.4 je vidět průběh učení - grafy ukazují závislost okamžité hodnoty advantage na časovém kroku - v prostředí *undulatory locomotion*. Zvýrazněná hodnot je upravena za pomoci exponenciálního klouzavého průměru s koeficientem 0.6. Poloprůhledná barva znázorňuje neupravenou hodnotu. Z grafů je zřejmé, že agent v průběhu učení procházel převážně stavy u okraje vymezené plochy, které s velkou pravděpodobností vedou k pádu robotu.

Simulace *sidewinding* nebyla provedena vůbec - proces učení pohybu vlněním v bočním směru autor považoval za složitější, a proto upřednostnil učení vlnění v přímém směru, doufajíc, že spíš přinese nějaké výsledky. Vzhledem k vyčerpání vymezeného času ovšem nezbývá než konstatovat, že proces učení nebyl úspěšný.



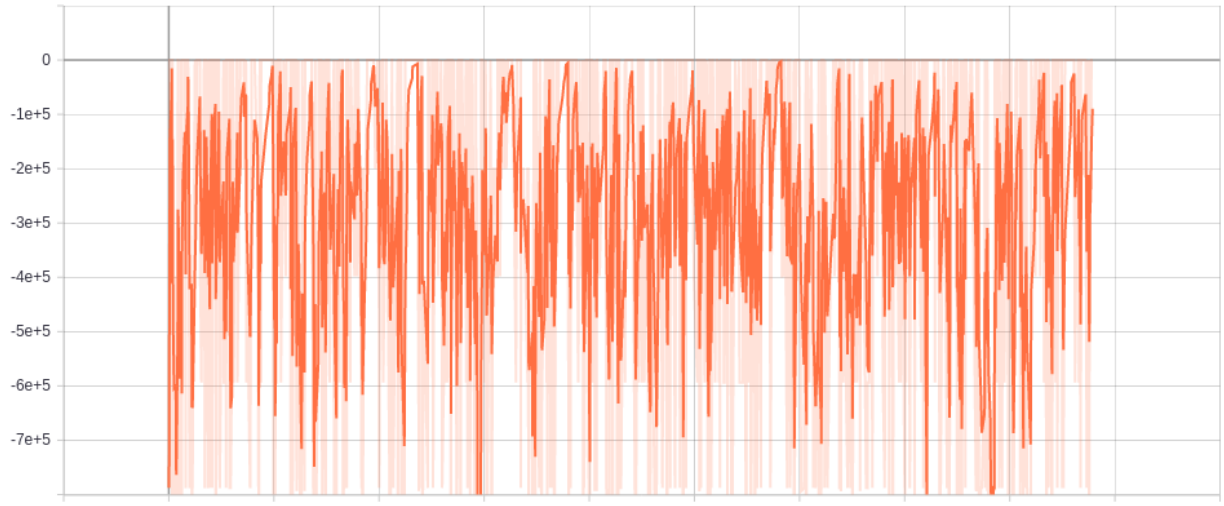
5 277 500 kroků

Obrázek 8.1: Graf závislosti advantage na čase (1/4)



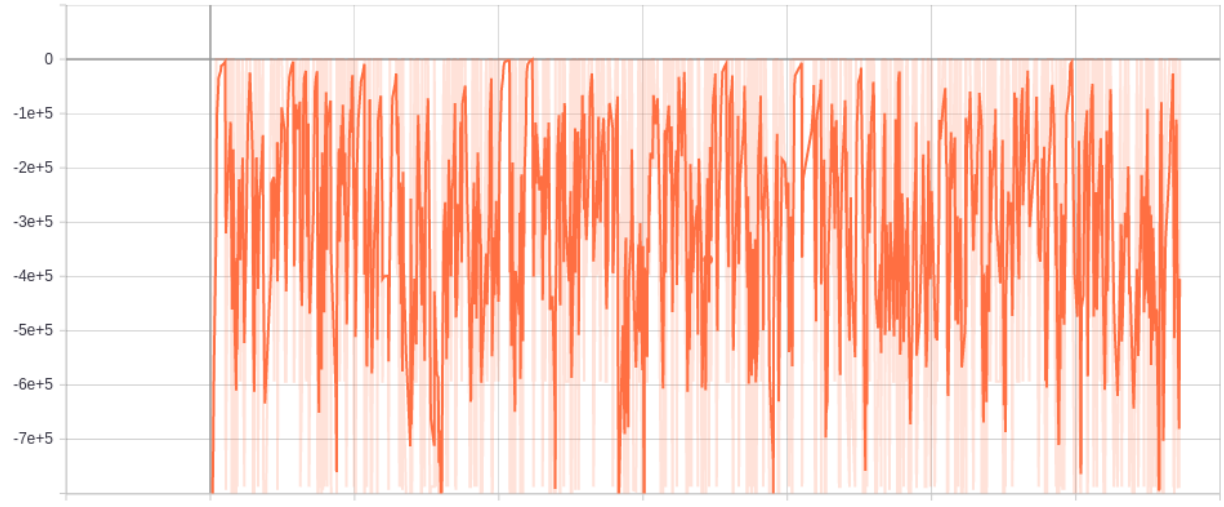
720 000 kroků

Obrázek 8.2: Graf závislosti advantage na čase (2/4)



905 000 kroků

Obrázek 8.3: Graf závislosti advantage na čase (3/4)



32 500 kroků

Obrázek 8.4: Graf závislosti advantage na čase (4/4)

9 Závěr

Diplomová práce pokrývá problematiku hlubokého posilovaného učení s použitím simulačního prostředí CoppeliaSim a knihoven tensorflow, openai.gym a stable_baselines. Autor vytvořil model robotu a navrhl pro něj simulační prostředí pro učení dvou typů hadovitých pohybů - pohyb vlněním v přímém a bočním směru s využitím výše zmíněných nástrojů.

Proces učení byl vyhodnocen jako neúspěšný a na základě toho se autor pokusil označit ty prvky systému, které mohly neúspěch způsobit. V této práci autor nastínil několik možných vylepšení představeného prostředí pro hluboké posilované učení. Další výzkum v této oblasti je rozhodně žádoucí, neboť celé odvětví hlubokého posilovaného učení je relativně nové.

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Použité zkratky a symboly

α	Učící krok, $\alpha \ll 1$
δ	TD-odchylka, rozdíl mezi TD-cílem a současnou hodnotou
γ	Slevový faktor, $\gamma \in (0; 1\rangle$, typicky blízko 1
$\mathbb{E}$	Očekávaná střední hodnota, často určena jako vážený průměr
$\mathbf{1}$	Binární indikátorová funkce, nabývá hodnoty 1, je-li splněna její podmínka, jinak 0
ε	Parametr hladovosti algoritmu, $\varepsilon \ll 1$
A3C	Asynchronous Advantage Actor-Critic - Asynchronní <advantage> actor-critic
D4PG	Distributed Distributional DDPG
DDPG	Deep Deterministic Policy Gradient - Hluboký deterministický gradient strategie
DQN	Deep Q Network, hluboká Q [neuronová] síť
MADDPG	Multi Agent DDPG
MC	Monte-Carlo
MRP	Markovovský rozhodovací proces
TD	temporal-difference

Seznam použité literatury

[1] HOLČÍK, Jiří, KOMENDA, Martin (eds.) a kol. *Matematická biologie: e-learningová učebnice* [online]. 1. vydání. Brno: Masarykova univerzita, 2015. ISBN 978-80-210-8095-9. [cit. 2020-06-20]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat–umela-inteligence–neuronove-site-jednotlivy-neuron–uvod-do-neuronovych-siti–historie-nn

[2] CHANDRA, Akshay L. *McCulloch-Pitts Neuron: Mankind's First Mathematical Model Of A Biological Neuron* [online]. 2018 [cit. 2020-06-20]. Dostupné z: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

[3] SINČÁK, Peter a Gabriela ANDREJKOVÁ. *Stručne o histórii NN* [online]. [cit. 2020-06-20]. Dostupné z: https://web.archive.org/web/20110908020448/http://neuron-ai.tuke.sk/cig/source/publications/books/NS1/html/node9.html

[4] HAGAN, Martin. *Perceptron Learning Rule* [online]. Stillwater (OK): Oklahoma State University [cit. 2020-06-20].
Dostupné z: https://web.archive.org/web/20100617194308/http://hagan.ecen.ceat.okstate.edu/4_Perceptron.pdf

[5] MINSKY, Marvin a Seymour PAPERT. *Perceptrons: An Introduction to Computational Geometry*. Cambridge (MA): MIT Press, 1969. ISBN 9780262630221.

[6] MEHROTRA, Kishan. *Elements of artificial neural networks*. Cambridge (MA): MIT Press, c1997. ISBN 978-0-262-13328-9.

[7] SINČÁK, Peter a Gabriela ANDREJKOVÁ. *Nekontrolované učenie na FF NN* [online]. [cit. 2020-06-20]. Dostupné z: https://web.archive.org/web/20110908014042/http://neuron-ai.tuke.sk/cig/source/publications/books/NS1/html/node50.html

[8] MACQUEEN, J. *Some methods for classification and analysis of multivariate observations* [online]. Los Angeles: University of California Press, 1967 [cit. 2020-06-20]. Dostupné z: https://projecteuclid.org/download/pdf_1/euclid.bsmsp/1200512992

[9] NIELSEN, Michael A. *How the backpropagation algorithm works* [online]. 2019 [cit. 2020-06-20]. Dostupné z: http://neuralnetworksanddeeplearning.com/chap2.html

[10] HARMON, Mance a Stephanie HARMONOVÁ. *Reinforcement Learning: A Tutorial* [online]. 17s. [cit. 2020-02-04]. Dostupné z: https://www.cs.toronto.edu/~zemel/documents/411/rltutorial.pdf

[11] WENG, Lilian. A (Long) Peek into Reinforcement Learning. *Lil'Log* [online]. 2018, 19.2.2018 [cit. 2020-02-04]. Dostupné z: https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

[12] WENG, Lilian. Policy Gradient Algorithms. *Lil'Log* [online]. 2018, 8.4.2018 [cit. 2020-02-04]. Dostupné z: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

[13] SUTTON, Richard a Andrew BARTO. *Reinforcement Learning: An Introduction*. 2. vyd. Cambridge (MA): MIT Press, 2017. Dostupné také z: http://www.incompleteideas.net/book/RLbook2018.pdf

[14] PATEL, Yash. Reinforcement Learning w/ Keras + OpenAI: Actor-Critic Models. *Towards Data Science: A Medium publication sharing concepts, ideas, and codes* [online]. 31.7.2017 [cit. 2020-02-18]. Dostupné z: https://towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69

[15] Reinforcement Learning - Part 3. *Mpatacchiola's blog* [online]. 2017, 29.1.2017 [cit. 2020-02-17]. Dostupné z: https://mpatacchiola.github.io/blog/2017/01/29/dissecting-reinforcement-learning-3.html

[16] KAPOOR, Sanyam. Policy Gradients in a Nutshell. *Towards Data Science: A Medium publication sharing concepts, ideas, and codes.* [online]. https://towardsdatascience.com/, 2.6.2018. [cit. 2020-02-17]. Dostupné z: https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d

[17] Van HASSELT, Hado, Arthur GUEZ a David SILVER. Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)* [online]. [cit. 2020-02-17]. Dostupné z: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12389/11847

[18] LILLICRAP, Timothy P., Jonathan J. HUNT, Alexander PRITZEL, Nicolas HEESS, Tom EREZ, Yuval TASSA, David SILVER a Daan WIERSTRA. *Continuous Control with Deep Reinforcement Learning* [online]. 5.6.2019. 14s. [cit. 2020-02-17]. arXiv: 1509.02971v6. Dostupné z: https://arxiv.org/pdf/1509.02971.pdf

[19] LOWE, Ryan, Yi WU, Aviv TAMAR, Jean HARB, Pieter ABBEEL a Igor MORDATCH. *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments* [online]. 16.1.2018. 16s. [cit. 2020-02-18]. arXiv:1706.02275v3. Dostupné z: https://arxiv.org/pdf/1706.02275.pdf

[20] BARTH-MARON, Gabriel, Matthew W. HOFFMAN, David BUDDEN, et al. *Distributed Distributional Deterministic Policy Gradients* [online]. 16.1.2018. 16s. [cit. 2020-02-18]. Dostupné z: https://openreview.net/pdf?id=SyZipzbCb

[21] SCHAUL, Tom, John QUAN, Ioannis ANTONOGLOU a David SILVER. *Prioritized Experience Replay* [online]. 21s. [cit. 2020-02-18]. arXiv:1511.05952. Dostupné z: https://arxiv.org/pdf/1511.05952.pdf

[22] JULIANI, Arthur. *Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C)* [online]. 17. 12. 2016 [cit. 2020-06-20]. Dostupné z: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

[23] MNIH, Volodymyr, Adrià P. BADIA, Mehdi MIRZA, Alex GRAVES, Tim HARLEY, Timothy P. LILLICRAP, David SILVER a Koray KAVUKCUOGLU. *Asynchronous Methods for Deep Reinforcement Learning* [online]. 19s. [cit. 2020-02-18]. arXiv:1602.01783v2. Dostupné z: https://arxiv.org/pdf/1602.01783.pdf

[24] KARAGIANNAKOS, Sergios. The idea behind Actor-Critics and how A2C and A3C improve them. *AI Summer* [online]. 17.11.2018 [cit. 2020-02-18]. Dostupné z: https://theaisummer.com/Actor_critics/

[25] Van WESEL, Perry a Alwyn GOODLOE. *Challenges in the Verification of Reinforcement Learning Algorithms* [online]. VI. 2017. 30s. [cit. 2020-02-04]. NASA/TM–2017–219628. Dostupné z: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20170007190.pdf

[26] BENNETT, Jake. The Algorithm Behind the Curtain: Reinforcement Learning Concepts (2 of 5). *Random Ant* [online]. Seattle, 19.5.2016 [cit. 2020-02-04]. Dostupné z: https://randomant.net/reinforcement-learning-concepts/

[27] BENNETT, Jake. The Algorithm Behind the Curtain: Understanding How Machines Learn with Q-Learning (3 of 5). *Random Ant* [online]. Seattle, 20.5.2016 [cit. 2020-02-04]. Dostupné z: https://randomant.net/the-algorithm-behind-the-curtain-understanding-how-machines-learn-with-q-learning/

[28] SILVER, David. *Lecture 2: Markov Decision Processes* [online]. 57s. [cit. 2020-02-05]. Dostupné z: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf

[29] MNIH, Volodymyr, Koray KAVUKCUOGLU, David SILVER, Alex GRAVES, Ioannis ANTONOGLOU, Daan WIERSTRA a Martin RIEDMILLER. *Playing Atari with Deep Reinforcement Learning* [online]. 9s. [cit. 2020-02-11]. Dostupné z: https://www.cs.toronto.edu/∼vmnih/docs/dqn.pdf

[30] Improvements in Deep Q Learning. *FreeCodeCamp.org* [online]. 5.7.2018 [cit. 2020-02-11]. Dostupné z: https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/

[31] Van HASSELT, Hado, Arthur GUEZ a David SILVER. Deep Reinforcement Learning with Double Q-learning [online]. 8.12.2015, 13s. [cit. 2020-02-11]. arXiv:1509.06461v3. Dostupné z: https://arxiv.org/pdf/1509.06461.pdf

[32] MNIH, Volodymyr. *Deep Q-Networks* [online]. 24s. [cit. 2020-02-11]. Dostupné z: https://drive.google.com/file/d/0BxXI_RttTZAhVUhpbDhiSUFFNjg/view

[33] MOON, Brad. *Snake locomotion* [online]. c2001 [cit. 2020-06-20]. Dostupné z: https://userweb.ucs.louisiana.edu/∼brm2286/locomotn.htm

[34] Orbito Palo. *How does a snake move without legs?* [online]. Utrecht, @2018 [c2018], 2. 1. 2019 [cit. 2020-06-20]. Dostupné z: https://orbitopalo.com/how-does-a-snake-move-without-legs/

[35] User:Borkin. *Undulatory locomotion* [online]. 2. 12. 2009 [cit. 2020-06-20]. Dostupné z: http://soft-matter.seas.harvard.edu/index.php/Undulatory_locomotion

[36] HILLEBRAND, Steve. *Eastern garter snake slithers through a muddy area* [online]. U.S. Fish and Wildlife Service [cit. 2020-06-20]. Dostupné z: https://commons.wikimedia.org/wiki/File:Eastern_garter_snake_slithers_through_a_muddy_area.jpg

[37] GONG, Chaohui, Ross HATTON a Howie CHOSET. *Conical sidewinding.* DOI: 10.1109/ICRA.2012.6225377 [online]. 2012 [cit. 2020-06-20]. Dostupné z: https://www.researchgate.net/publication/254041447_Conical_sidewinding

[38] Jak se pohybují hadi. *Biologie kolem nás* [online]. 27. 1. 2015 [cit. 2020-06-20]. Dostupné z: http://svet-biologie.blog.cz/1501/jak-se-pohybuji-hadi

[39] KARAGIANNAKOS, Sergios. *The idea behind Actor-Critics and how A2C and A3C improve them.* The AI Summer [online]. 17. 11. 2018 [cit. 2020-06-24]. Dostupné z: https://theaisummer.com/Actor_critics/